基于 GPU 实现叠前时间偏移走时计算的并行算法①

张　清[1]，迟旭光[3]，谢海波[1]，赵开勇[1,2]，吴　庆[1]，陈　维[3]，王狮虎[3]，褚晓文[1,2]

[1](浪潮集团高效能服务器和存储技术国家重点实验室，北京 100085)
[2](香港浸会大学 计算机系，香港)
[3](中国石油集团东方地球物理勘探有限责任公司，涿州 072751)

摘　要：走时计算是叠前时间偏移计算中最耗时的部分，通过分析传统的串行走时算法，发现静态 8 点插值算法非常适合在 GPU 上运行。首先利用 CUDA 技术对静态 8 点插值算法进行并行化改造，设计静态 8 点并行插值算法，然后测试其正确性，统计其相对误差情况。实验表明此算法比工业生产上的动态插值算法更准确，最后我们利用体偏作性能测试。试验结果表明，运行在 GPU 上的静态 8 点并行插值算法内核性能是运行在 CPU 上的动态插值算法内核的 22.76 倍。这说明，静态 8 点并行插值算法适合进行走时计算，并且可以应用于工业生产上。

关键词：叠前时间偏移；地震勘探；图形处理器；计算统一设备架构；走时计算；并行计算

Parallel Algorithm Based on the Travel Time Computing of Pre-stack Time Migration Using GPU

ZHANG Qing[1], CHI Xu-Guang[3], XIE Hai-Bo[1], ZHAO Kai-Yong[1,2], WU Qing[1], CHEN Wei[3], WANG Shi-Hu[3], ZHU Xiao-Wen[1,2]

[1](The State Key Laboratory of High Performance Server and Storage Technology, Inspur Group, Beijing 100085, China)
[2](Department of Computer Science, Hong Kong Baptist University, Hongkong, China)
[3](BGP , Zhuozhou 072751, China)

Abstract: The computation of traveling time of Pre-Stack Time Migration (PSTM) is the most time-consuming part of whole procedure. In this paper, we present a novel parallel algorithm to low the time of traveling-time computation. By using static interpolation with step of 8 points instead of dynamic step length, the new designed algorithm performs more accuracy and performance. Experimental results show that the speed-up number reaches 22.76 times. The statistical relative error demonstrates the better accuracy comparing with the commercial codebases.

Key words: PSTM; seismic exploration; GPU; CUDA; the travel time computing; parallel computing

1　引言

1.1 叠前时间偏移走时计算

地震资料处理是石油和天然气勘探开发领域中非常重要的环节。传统的方法是通过人工放炮方式产生地震波，通过地面检波器将地下不同地质层反射回来的地震波信号收集后，利用大型计算机通过多套专业处理软件和一套完整的叠前时间偏移、叠前深度偏移软件系统进行资料处理，从而得到地下的构造以及成像。为石油钻井提供更加可靠的勘探数据，用于勘探专家进行下一步的分析和解释，掌握地下的油气构造[1,2]。

叠前时间偏移是构造复杂成像最有效的方法之一，它能适应纵向速度变化较大的情况，适用于大倾角的偏移成像[3,4]。而其中走时计算是最关键、最费时的一个环节[5]，其采用的算法大致分为三类，即直射线、弯曲射线以及非对称走时计算[6]。由于叠前时间偏移动辄需要处理数以 TB 的海量数据，以串行方式实现的走时计算算法，处理性能十分低。鉴于此，本文基于 Kirchhoff 积分法叠前时间偏移，采用 GPU 编程实现并行走时计算算法，其处理性能获得大大提升。

① 收稿时间:2010-11-11;收到修改稿时间:2010-12-16

1.2 GPU/CUDA 技术介绍

GPU(Graphic Processing Unit，图形处理器)，由于其拥有数十倍于 CPU 的浮点运算能力，其作为相较 FPGA 等更为非常成熟的商业芯片，从而表现出更好的性价比[7]。从 2000 年左右开始，有部分研究者开始采用 GPU 强大的图像处理能力来进行通用计算。传统的 GPGPU（General-Purpose computation on Graphics Processing Units）的处理方法是把通用的计算问题转换为 GPU 中能处理的图形问题，然后通过图形编程语言来进行编程。这样的编程模式使得开发难度很高，开发者必须熟练地掌握图形编程能力[8]。

CUDA（Compute Unified Device Architecture，统一计算架构）是由 NVIDIA 所推出的一种集成技术，它是针对 GPU (graphic processing unit, 图形处理器) 的并行开发环境[9]。利用 CUDA 技术,开发人员能够利用 GPU 处理极复杂的密集计算难题。因此它在石油天然气地震处理、金融风险管理、产品设计、媒体图像以及科学研究等领域具有重要应用。例如, Samer AI-Kiswany 等人设计了一个 StoreGPU 中间件,通过 GPU 处理器来加速处理分布式存储系统。Lessig 在 CUDA 平台上实现了 MRRR 算法,与 LAPA2CK 相比实验性能得到 50 倍的提升[10,11]。在科学计算方面, Tolke[12]发现可利用 GPU 来处理 2D-Lattice Boltzmann 算法; Acceleware 公司应用 GPU 来处理电磁模拟、地震数据处理和图像重构。

本文提出基于 GPU 的叠前时间偏移走时计算并行算法。我们首先分析叠前时间偏移走时计算的串行算法，提出适合 GPU 运行的走时计算并行算法，并采用 CUDA 技术实现此算法；最后分析其相对误差，并测试其性能。

2 叠前时间偏移走时计算相关串行算法介绍

地震波从震源出发，当遇到不同介质时，发生折射，然后传回观测点。我们把折射点称为成像点，若干个成像点组成一个类似长方体的三维结构的成像空间。叠前时间偏移走时计算是计算地震波从震源传到观测点所经过的时间，即计算地震波从震源传到成像点所经过的时间与地震波从成像点传到观测点所进过的时间之和。其中计算地震波从震源传到成像点所经过的时间和计算地震波从成像点传到观测点所进过的时间都称为走时计算。对于成像空间中的每一个成像点而言，需要计算不同震源到它的走时及它到相应观测点的走时，其走时计算，可通过两种方式实现：(1)直接根据公式精确计算；(2)依赖临近点进行线性插值计算。

2.1 无插值算法

无插值算法的定义是：某一点的走时计算结果并不通过插值方式计算得到，而是通过精确计算得出成像空间中每一点的走时，此算法是叠前时间偏移走时计算最为精准的算法。

2.2 静态 8 点插值算法

静态 8 点插值算法是针对无插值算法走时计算性能低而提出的，我们把成像空间中垂直地面方向，即深度方向上每相邻的 32 个点看成一层，索引值为 0 到 31，首先计算出索引值能被 8 整除的点的走时，即第 0、8、16、24 点的走时，第 1 至 7 点的走时通过第 0 和第 8 点线性插值得到，其它点以此类推，其算法插值的幅度为每 8 个点，并且幅度是不变的，故名静态 8 点插值算法，其具体实现步骤如下：

(1) 精确计算出索引值能被 8 整除的点的走时，并把结果存储起来；

(2) 判断一层（32 个点）内索引号不能被 8 整除点落在哪两个最近的能被 8 整除的点之间，并计算插值因子；

(3) 插值计算层内索引号不能被 8 整除的点的走时。

2.3 动态插值算法

为进一步提高走时计算性能，动态插值算法根据插值的正确误差范围内可以进行 8 点、16 点、24 点，最大可达到 32 点的动态插值，插值方式并一定固定为 8 点，可以通过判断动态选择插值方式，此算法不已在工业生产中被应用。

2.4 算法比较

以上三种算法中，无插值算法的结果最为精准，其他算法计算结果可以参照它作为比对基准。动态插值算法根据插值情况动态选择 8 点、16 点、24 点、32 点插值，它的计算性能是最佳的，但是它的判断分支较多，利用 CUDA 改造成并行算法后，在 GPU 上运行其性能会受到影响；而静态 8 点插值算法，因为不存在动态选择的分支，便于移植到 GPU 上运行，其性能会提高得更明显。所以我们基于静态 8 点插值算法

思想，采用 CUDA 技术，对此算法进行并行化设计，实现走时计算的静态 8 点并行插值算法。

3　基于GPU的静态8点并行插值算法设计

3.1 算法设计

对成像空间建立三维 X-Y-Z 坐标系，整个空间的点按照（x , y , z）坐标唯一确定，X、Y 坐标相同的点构成一条与 Z 轴平行的线，物理上是一条垂直于地面的竖线，将这条线上的连续的 32 个点划分为一层，基于静态 8 点插值算法思想，对于同一条线上每一个点的走时 T 并不需要精确计算出，而是首先并行计算出索引值能被 8 整除的点的走时 T，其余点的走时 T 依赖于与它临近的已精确计算的两个点的值作线性插值得到，其基于 GPU 的具体设计过程如下：

(1) 如图 1 所示，以 8 层为一个计算单元，1 个 warp 内 32 个线程精确并行计算出索引值能被 8 整除的 33 个点的走时 T，其中线程 0 多计算一个点；

(2) 判断一层内所有点的索引号落在哪两个最近的索引号能被 8 整除的点之间，然后计算其插值因子；

(3) 并行进行静态 8 点插值计算，如图 2 所示，以一层为一个计算单元，1 个 warp 内 32 个线程并行插值计算一层内所有点（包括索引号能被 8 整除的点和索引号不能被 8 整除的点）的走时 T，循环 8 次后，整个 8 层的所有点的走时 T 将被计算完成。

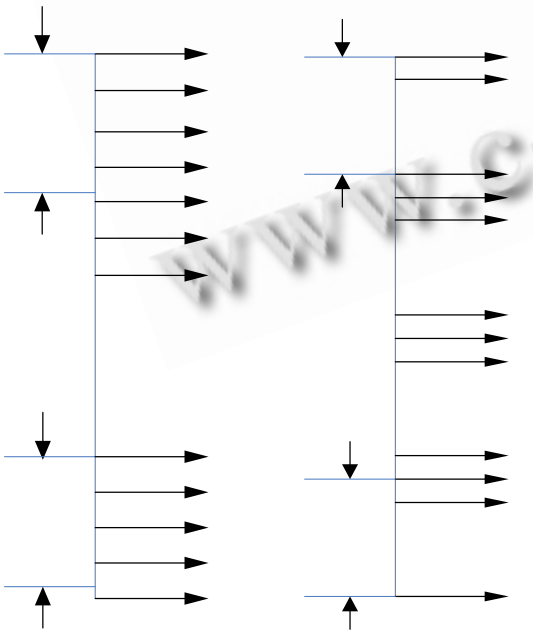

图 1 8 层内 32 线程执行图　图 2 1 层内 32 线程执行图

3.2 CUDA 线程模型设计

线程模型是用来明确 CUDA 程序内核的执行配置，根据 GPU 硬件资源，如寄存器个数、内存等，来定义网格和线程块，好的线程模型会使程序性能大大提高。

(1) 网格（grid）的定义：即把所有线程如何进行分块，定义线程块数和线程块的组织方式，这里我们根据成像空间的 X-Y 平面来划分线程块，其具体定义为 dimGrid(NX , NY / 10)；

(2) 线程块（block）的定义：即定义一个线程块有多少个线程和线程的组织方式，根据内核所需的寄存器数和共享内存数量，我们定义一个线程块为 320 个线程，其具体定义为 dimBlock(32 , 10)；

(3) 线程模型描述：grid 和 block 都是定义为二维的，线程总数为 NX * NY * 32，线程块数为(NX * NY) /10，每个 block 的 320 个线程处理 10 个 x ，y 坐标所对应的 Z 轴的所有点，即处理 Z 轴的 10 条如果把每个 block 的 320 个线程按照 32 个线程进行分组，每一组为一个 warp，那么整个 block 有 10 个 warp，第 0 个 block 的第 0 个 warp，即 threadIdx.y 等于 0 的线程处理第 0 个 x,y 坐标对应的 Z 轴的第一条线；第 0 个 block 的第 1 个 warp，即 threadIdx.y 等于 1 的线程处理第 1 个 x,y 坐标对应的 Z 轴的另一条线，如此类推，第 0 个 block 的第 9 个 warp，处理第 9 个 x,y 坐标对应的 Z 轴的一条线。同理其它 block 处理另外 10 个 x , y 坐标对应的 Z 轴的 10 条线。10 条线并行进行走时计算，可以使计算更均衡，性能更高。

3.3 CUDA 内存使用设计

根据 8 点静态并行插值算法、数据访问特点及 GPU 内存资源特性，选择不同的内存存放不同的数据，以达到性能最优。

(1) Global memory 使用：成像空间数据存储方式是先按照 Z 方向连续，同时 Z 方向维度大小总是 32 的整数倍。Z 方向的计算总是以 32 为单位进行计算。因此在同一时刻，half warp 线程可以同时访问成像空间的 16 个点，16 点在内存中是连续的，并且可以做到线程与访问点的一一对应，从而形成对 Global Memory 的合并访问，提高访存性能。

(2) Texture memory 使用：由于 GPU 中的 Texture memory 有 cache，把输入道数据、速度场等只读的频繁访问的大数据存放其中，将提高访存性能。

(3) Shared memory 使用：由于 Shared memory 为 GPU 的片上内存，访问速度快，对于一个 Block 块中公共的数据，如能被 8 整除的点的走时，可以放入共享内存中，将提高访存性能。

(4) Constant memory 使用：对于 PSTM 计算中只读的、频繁被访问的小数据块可以放入 Constant memory 中，将大大提高访存性能。

4 静态8点并行插值算法误差分析

我们选用运行在 CPU 上的无插值算法作为计算结果正确性比对的基准算法，分别比对工业上应用成熟的运行在 CPU 平台上的动态插值算法和运行在 GPU 上的静态 8 点并行插值算法的结果误差情况。

4.1 相对误差法

我们记运行在 CPU 上的无插值算法的输出的每一点的成像空间的输出值（以下简称 WOT 值）为 WOT_Base；运行在 GPU 上静态 8 点并行插值算法的输出的每一点的 WOT 值为 WOT_8_Static；运行在 CPU 上的动态插值算法输出的每一点的 WOT 值为 WOT_Dynamic。那么静态 8 点并行插值算法的输出的每一点的相对误差 = （|WOT_8_Static| - |WOT_Base|）/ |WOT_Base|；动态插值算法的输出的每一点的相对误差 = （|WOT_Dynamic| - |WOT_Base|）/ |WOT_Base|，相对误差越接近 0，说明算法越精准，以下通过测试进行统计分析静态 8 点并行插值算法与动态插值算法的误差情况。

4.2 相对误差统计结果

分别对运行在 CPU 上的无插值算法、运行在 GPU 上的静态 8 点并行插值算法以及运行在 CPU 上的动态插值算法输入 1000 道地震道数据进行叠前时间偏移计算的测试，从运行的结果中采样一些点，进行分析相对误差，其相对误差统计结果和两种算法相对误差比对图分别如表 1、图 3 所示。

4.3 结果分析

从图 3 可以看出，运行在 GPU 上的静态 8 点并行插值算法进行 PSTM 计算所得出的结果中相对误差为 0 的点数比动态插值算法多，但在其他误差区间中的点数都比动态插值算法都少，可以充分说明运行在 GPU 上的静态 8 点并行插值算法进行 PSTM 计算的结果比运行在 CPU 上的动态插值算法进行 PSTM 计算的结果要准确。

表 1 1000 道地震道数据测试相对误差比较表

PSTM 相对误差分析算法	静态 8 点并行插值算法	动态插值算法
PSTM 基准算法	无插值算法	无插值算法
运行平台	GPU	CPU
计算地震道道数	1000	1000
结果采样点数	30602	30602
相对误差为 0 的点数	22154	16851
相对误差为 0~10^{-6} 的点数	1522	1902
相对误差为 10^{-6}~10^{-5} 的点数	1	4
相对误差为 10^{-5}~10^{-4} 的点数	1110	1586
相对误差为 10^{-4}~10^{-3} 的点数	273	439
相对误差为 10^{-3}~10^{-2} 的点数	561	795
相对误差为 10^{-2}~10^{-1} 的点数	1173	1885
相对误差为 10^{-1}~1 的点数	2184	3836
相对误差为大于等于 1 的点数	1624	3304

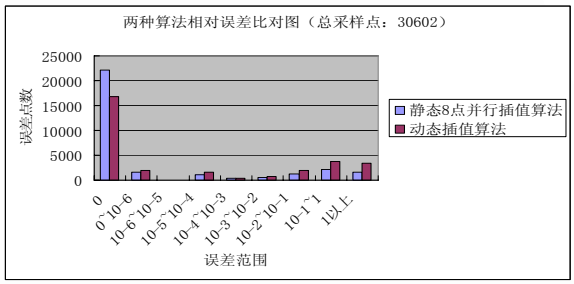

图 3 1000 道地震道数据测试相对误差比较图

5 性能测试

5.1 测试环境及数据

测试环境包括硬件环境、软件环境、叠前时间偏移运行内核，其中叠前时间偏移运行内核为叠前时间偏移的核心计算部分，即包括走时计算部分；测试数据为输入的测试地震道数据集。对于成像空间而言，它是四维的，其中第一维为 X 轴方向的大小；第二维为 Y 轴方向的大小；第三维为炮点与接收点的偏移范围大小；第四维为 Z 轴方向的大小，具体各项参数如表 2 所示。

表 2 测试环境和测试数据

硬件环境	CPU：Intel Xeon 四核 E5620（Westmere，2.4GHz）
	内存：6×4GB DDR3 ECC REG 1333 内存
	GPU：Tesla C1060
软件环境	OS: 64 位 Linux RedHat 4.5
	编译器: ICPC
	编译选项: O2

叠前时间偏移运行内核	运行在 CPU 上的动态插值算法内核	
	运行在 GPU 上的静态 8 点并行插值算法内核	
成像空间	第一维 NX 大小	50
	第二维 NY 大小	1097
	第三位 NOFF 大小	1
	第四位 NZ 大小	1504
输入道集	输入道数：450000	
	道长：6000ms	

5.2 性能结果

为了保证测试性能结果的稳定性，我们对上述线偏和体偏作业分别进行了 10 次测试，并且动态插值算法运行在 CPU 平台上，静态 8 点并行插值算法运行在 GPU 平台上，其性能结果如下表 3 所示。

表 3　两种算法性能结果比对表

运行在 CPU 上的动态插值算法内核运行时间	54320s
运行在 GPU 上的静态 8 点并行插值算法内核运行时间	2383s
内核加速比	22.76 倍

6　结论

我们通过分析走时计算的相关串行算法，设计适合 GPU 运行的静态 8 点并行插值算法，并通过相对误差测试，静态 8 点并行插值算法在准确度上要优于工业生产所采用的动态插值算法，其性能也比动态插值算法要好，加速比达到 22.76 倍。这说明，静态 8 点并行插值算法精确度高，性能优越，适合进行叠前时间偏移走时计算，对油气勘探领域地震资料处理有重大意义。

参考文献

1 王余庆,等.叠前偏移技术探讨及应用.西北油气勘探,2006,18(2):31-39.

2 王棣等.叠前时间偏移方法综述.勘探地球物理进展,2004,27(5):313-320.

3 王华忠,等.起伏地表条件下偏移到多偏移距叠前时间偏移.勘探地球物理进展,2007,(5):361-367.

4 刘洪,等.基于横向导数的走时计算方法及其叠前时间偏移应用.石油物探,2009,48(1):3-10.

5 Wiggins W. Kirchhoff integral extrapolation and migration of nonplanar data. Geophysics, 1984,49(8):1239-1248.

6 李博等.地震叠前时间偏移的一种图形处理器提速实现方法.地球物理学报,2009,52(1):245-252.

7 张浩,等.GPU 的通用计算应用研究.计算机与数字工程,2005,33(12):60-62.

8 王凯,等.GPU 通用计算在 LBM 方法中的应用.计算机工程与设计,2009,30(19):4513-4515.

9 郭境峰,等.新一代高性能运算技术——CUDA 简介.现代科技,2009,8(6):29-30.

10 LESSIG C. An implementation of MRRR algorithm on a data-parallel coprocessor. Toronto: University of Toronto, 2008.126-129.

11 吴连贵,等.基于 CDUA 的地震数据相干体并行算法.计算应用,2009,(3):294-296.

12 TOLKE J. Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA. Proc. of Conference on Computing Frontiers. 2008.8-12.